Proof Search in Acyclic Matrix Graphs (Algorithms and Theory of Computing)

AUTHOR(S):

Fronhofer, Bertram

数理解析研究所講究録
1041 巻 1998 年 87-94

Proof Search in Acyclic Matrix Graphs

Bertram Fronhöfer*
Meme Media Laboratory, Hokkaido University, Sapporo 060 Japan
fronhoef@informatik.tu-muenchen.de

Abstract

In [2] we gave a characterization of theoremhood in terms of the Connection Method for the multiplicative fragments of Affine (Contraction-Free) and Linear Logic. Such matrix characterisations constitute a basis for the adaption of the classical path checking procedures to the just mentioned logics.
In this paper we establish computationally advantageous variants of the previously developed matrix characterisations. Instead of the complementarity of paths they hinge on the total connectedness of a sufficient number of virtual columns in a matrix (non-purity). From these refined matrix characterisations we derive a pruning technique for proof search in acyclic (and linear) matrix graphs, which can be used for multiplicative Affine Logic. Extending proof search to multiplicative Linear Logic, we show that the additionally required minimality of spanning sets is assured by depth-first search, and that the necessary connectedness of all literals can be tested incrementally.
This paper is a shortened version of [3] which can be obtained from the author.

1 Introduction

Classical logic, when defined with *Sequent Calculus*, is determined by the availability of the structural rules of contraction, weakening and exchange. Dropping (or weakening) some of these structural rules yields so-called *substructural logics*. In the absence of contraction or weakening, different equivalent ways of introducing into sequents the connectives $\wedge$, $\vee$ and $\longrightarrow$ are no longer equivalent, and we must distinguish the *multiplicative* connectives $\otimes$, $\parr$ and $\multimap$ from their *additive* counterparts $\&$, $\oplus$ and $\twoheadrightarrow$. The multiplicative fragment, given by the multiplicative connectives and negation, is conceptually simpler than the additive one, and efficient proof search seems to be more feasible.
One of the approaches to proof search for classical logic is the Connection Method (see [1] and Section 2 below). With this approach formulae are transformed into so-called matrices, connections are added, and there is the fundamental theorem that a formula is a theorem of classical propositional logic iff all paths through the respective matrix contain a connection. (A set of connections which makes all paths complementary is called spanning.) In order to characterize also theoremhood in substructural logics by means of matrices and connections further conditions must be demanded. In [2] we showed that for characterizing theoremhood in the multiplicative fragment of Affine Logic, in addition to complementarity the two further conditions linearity and acyclicity must be fulfilled (see Theorem 1 below). This leads in a straightforward manner to a rather trivial proof procedure for multiplicative Affine Logic: We take a proof procedure based on the classical Connection Method, which enumerates all spanning sets of connections, and check incrementally linearity and acyclicity of the spanning sets under construction.

*On leave from: Institut für Informatik, TU München, D-80290 München

However, we discovered that our acyclicity condition is very strong and allows to replace complementarity by the weaker condition of *non-purity* which is also easier to check (Def. 1 below): Instead of caring about complementarity of paths we only have to look for a set of connections which connect each literal in a sufficient (saturated) amount of virtual columns of the matrix. This leads to an improved proof search procedure, because it allows to refrain from checking for complementarity all continuations of a partial path through a literal L, if this literal L is already connected.
The proof procedure obtained so far can be applied in a naïve way to the multiplicative fragment of Linear Logic by checking in addition whether the generated sets of connections are also minimal spanning and connect all literals (total connectedness). Fortunately, with depth-first search, the developed proof procedure generates only minimal spanning sets of connections. Consequently, an additional, and in general quite costly, minimality check becomes superfluous. Moreover, it turns out that the test of total connectedness can be done incrementally during proof search, which permits an earlier detection of eventually failing proof attempts. Thus we achieve exponential gains over related methods.
This paper is a very shortened verison of [3] to which we refer for more details and for proofs of the lemmata presented in the sequel.

2 The Connection Method

We briefly present the substructural logics which will be dealt with in the following and shortly review the main concepts of the Connection Method. We assume some familiarity with the Connection Method for classical logic [1] and mainly point out what is different for its application to the multiplicative fragments of substructural logics like Affine or Linear. For more details we refer the reader to [2].
By the ***multiplicative fragment*** we understand all well-formed formulae built from propositional variables and the connectives $\multimap$ (implication), $\otimes$ (conjunction), $\parr$ (disjunction) and $\neg$, for which we assume the multiplicative sequent rules. Moreover, we consider the structural rules of weakening, exchange and contraction. Dropping the rules of contraction we get ***multiplicative Affine Logic***. If we drop the rules of weakening as well, then we get ***multiplicative Linear Logic***.
For characterizing Affine Logic by means of the Connection Method a combined generalisation of its concepts into the following three directions had to be developed for reasons which will become clear in the sequel:

1. Non-normal form matrices,
2. multi-occurrences of literals and
3. relativisation to arbitrary sets of connections.

A ***(clause) normal form matrix*** is a set (***row***) of sets (***columns***) of literals. ***Literals*** are propositional variables or their negations. In matrices we denote negation by overlining. Without contraction the transformation of a formula into a normal form matrix is no more possible because we lack the necessary distributive laws between $\otimes$ and $\parr$. We just achieve ***negation normal form***, i.e. a formula built from negated or unnegated propositional variables with the connectives $\otimes$ and $\parr$. This corresponds to ***non-normal form matrices*** (*point* (1) *above*). In contrast to normal form matrices, they have arbitrarily deep nestings, i.e. they are a set of sets of sets ..., respectively a ***row*** of ***columns*** of ***rows*** ... The symbol $\in$ denotes set membership and $\Subset$ denotes its reflexive transitive closure. Rows or columns $\mathcal{N}$ with $\mathcal{N} \in \mathcal{N}'$ are called ***top-level elements*** of $\mathcal{N}'$.

Without contraction we also loose the idempotence of conjunction ($\otimes$) and disjunction ($\parr$). This requires to ***distinguish different occurrences*** L^1 ***and*** L^2 ***of a (logical) literal*** within the same row or column $\mathcal{N}$ of a matrix $\mathcal{M}$ ($L^1, L^2 \in \mathcal{N} \Subset \mathcal{M}$). In addition, we have to assure during the translation of a formula into a matrix, that different atomic subformulae will yield different occurrences of literals in the matrix (*point* (2) *above*).

A ***path*** $\mathfrak{p}$ ***in a row*** $\mathcal{N} = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ is the union of paths $\mathfrak{p}_1$ ***in*** $\mathcal{M}_1$, ..., $\mathfrak{p}_n$ ***in*** $\mathcal{M}_n$ with $\mathcal{M}_i \in \mathcal{C}_i$ or $\mathcal{M}_i = \mathcal{C}_i$ in the case of $\mathcal{C}_i$ being a literal. If $\mathcal{C}_i$ or $\mathcal{M}_i$ is an occurrence L^1 of a literal, then the path $\mathfrak{p}_i$ is $\{L^1\}$.

Two (logical) literals K and L are called ***complementary***, if there is a propositional variable P and $K = P$ and $L = \overline{P}$ or vice versa. Since for an arbitrary formula F holds that $\neg\neg F$ is equivalent to F, we make the convention that $\overline{K}$ denotes the literal being ***complementary*** to the literal K, where K may be a propositional variable or its negation. A ***connection*** in a matrix $\mathcal{M}$ is an unordered pair of occurrences of complementary literals $L \Subset \mathcal{M}$ and $K \Subset \mathcal{M}$, denoted by (L, K) or (K, L).

A path $\mathfrak{p}$ in a matrix $\mathcal{M}$ is a ***complementary path*** with respect to a set of connections $\mathfrak{S}$, if $\mathfrak{p}$ ***contains a connection*** of $\mathfrak{S}$, i.e. there are occurrences of literals $L \Subset \mathcal{M}$ and $K \Subset \mathcal{M}$ with $(K, L) \in \mathfrak{S}$ and $K, L \in \mathfrak{p}$. A set of connections $\mathfrak{S}$ is called ***spanning*** for a matrix $\mathcal{M}$ iff every path in $\mathcal{M}$ contains at least one connection from $\mathfrak{S}$.

For classical logic holds that if there is a ***spanning set of connections*** for a matrix, then also the set of all connections is spanning. In the following we have to state more explicitly for a given matrix which set of connections we are dealing with (*point* (3) *above*). This leads to the concept of a ***matrix graph*** $(\mathcal{M}, \mathfrak{S})$, i.e. a pair consisting of a matrix $\mathcal{M}$ and a particular subset $\mathfrak{S}$ of the set of all connections. A ***complementary matrix graph*** is a matrix graph $(\mathcal{M}, \mathfrak{S})$ such that $\mathfrak{S}$ is a spanning set of connections for $\mathcal{M}$.

For a matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathcal{M} \equiv \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$, given subsets $\mathcal{D}_i \subset \mathcal{C}_i$, and $\mathcal{N} := \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, we define the ***strong restriction of*** $\mathfrak{S}$ ***to*** $\mathcal{N}$ as

$$\mathfrak{S}|_{\mathcal{N}} := \{\, (K, L) \in \mathfrak{S} \mid K \Subset \mathcal{N} \text{ and } L \Subset \mathcal{N} \,\}$$

A matrix graph $(\mathcal{M}, \mathfrak{S})$ is called ***path minimal*** iff for every connection in $\mathfrak{S}$ exits a path which contains no further connection from $\mathfrak{S}$. A matrix graph $(\mathcal{M}, \mathfrak{S})$ is ***minimal complementary*** iff it is complementary and path minimal.

A literal in a matrix graph $(\mathcal{M}, \mathfrak{S})$ is ***connected*** iff it is part of a connection in $\mathfrak{S}$. Otherwise it is ***unconnected***. A part of a matrix graph—e.g. a row, column, path, etc.—is ***connected*** iff it contains a connected literal, it is ***totally connected*** iff all its literals are connected, and it is ***isolated*** iff none of its literals is connected.

Virtual columns are defined as the vertical analogues of (the horizontal) paths.

A part $\mathcal{N}$ of a matrix is ***positive/negative*** iff each literal $L \Subset \mathcal{N}$ is a propositional variable resp. the negation of a propositional variable. A virtual column $\widetilde{\mathcal{C}} \Subset \mathcal{M}$ is called ***relevant in*** $\mathcal{M}$ if there is a minimal spanning set $\mathfrak{S}$ for $\mathcal{M}$ such that $\widetilde{\mathcal{C}}$ is totally connected by $\mathfrak{S}$.

Theorem 1 ([2, Theorem 35]) *Given a multiplicative formula F and $\mathcal{M}$ its translation into a non-normal form matrix as discussed above:*

F is a theorem of Affine Logic iff there is a linear, acyclic and complementary matrix graph $(\mathcal{M}, \mathfrak{S})$.

F is a theorem of Linear Logic iff there is a linear, totally connected, acyclic and minimal complementary matrix graph $(\mathcal{M}, \mathfrak{S})$.

Linearity of a matrix graph $(\mathcal{M}, \mathfrak{S})$ means that each literal in $\mathcal{M}$ may be involved in

at most one connection of $\mathfrak{S}$, while ***total connectedness*** means that each literal in $\mathcal{M}$ must be involved in at least one connection of $\mathfrak{S}$. More difficult to explain is ***acyclicity***. Since in the following it will not be more than a black box, we skip its definition for reasons of space and refer to [3] or [2].

3 Non-Pure Matrix Graphs

If we add conditions like linearity and acyclicity to complementarity, it is interesting to analyse to which degree these conditions are independent, because replacing some of them by weaker, but equivalent ones, might facilitate proof search due to simpler checks. Indeed, it turns out that under rather elementary assumptions on a set of connections, complementarity is entailed by acyclicity. It suffices to require a suitable generalization to non-normal form matrices of the concept of pure literals, and this in addition with respect to a considered set of connections (Lemma 2 below). First some definitions.

Definition 1 *A column $\mathcal{C} \Subset \mathcal{M}$ of a matrix graph $(\mathcal{M}, \mathfrak{S})$ is **pure** iff there is an isolated path in $\mathcal{C}$. A matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathfrak{S} \neq \emptyset$ is **non-pure** iff equivalently: **Either** there is no connected pure column in $(\mathcal{M}, \mathfrak{S})$ **or** for every literal $L \Subset \mathcal{M}$ which is connected in $\mathfrak{S}$ exists a virtual top-level column $\widetilde{\mathcal{C}} \widetilde{\in} \mathcal{M}$ which is totally connected in $\mathfrak{S}$. A matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathfrak{S} \neq \emptyset$ is called **minimal non-pure** iff there is no nonempty subset $\mathfrak{S}' \subset \mathfrak{S}$, $\mathfrak{S}' \neq \mathfrak{S}$, with $(\mathcal{M}, \mathfrak{S}')$ non-pure.*

Lemma 2 *If a matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathfrak{S} \neq \emptyset$ is acyclic and non-pure, then it is complementary.*

If a matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathfrak{S} \neq \emptyset$ is minimal non-pure and acyclic, then it is minimal complementary.

On the basis of the investigations of this section, Theorem 1 can be rephrased as follows:

Theorem 3 *Given a multiplicative formula F and $\mathcal{M}$ its translation into a matrix, then we get the following refinement of Theorem 1:*

F is a theorem of Affine Logic iff there is a linear, acyclic, non-pure matrix graph $(\mathcal{M}, \mathfrak{S})$ with $\mathfrak{S} \neq \emptyset$.

F is a theorem of Linear Logic iff there is a linear, acyclic, totally connected, minimally non-pure matrix graph $(\mathcal{M}, \mathfrak{S})$.

4 Generating Spanning Sets

In Theorem 1 we characterized theorems of multiplicative Affine Logic by complementary matrix graphs which fulfil in addition the conditions linearity and acyclicity. On this basis a proof procedure for multiplicative Affine Logic can be obtained in an extremely straightforward way as follows: (1) We take a path checking procedure for classical propositional logic which guarantees to generate *at least all minimal spanning sets of connections* for a given matrix. (2) For every spanning set returned by this procedure, we test the properties linearity and acyclicity. (3) Since linearity and acyclicity of sets of connections are inherited by subsets, we finally improve this procedure by performing the linearity and acyclicity check incrementally (whenever a new connection is made), instead of waiting till an entire spanning set has been constructed.

For the enumeration at least of all minimal spanning spanning sets of connections the path checking procedure given in [1] for first-order logic is a good starting point Adapting it to non-normal form matrices yields the proof procedure SSC (***S***panning ***S***et ***C***onstruction)

for classical propositional logic (Definition 2).

Definition 2 (***proof procedure*** SSC) *This proof procedure is given in Figure 1.* $\mathcal{M}$ *refers to the given matrix whose spanning sets shall be determined.* S *is the set of connections under construction. There is a task pool with the operations* ***put*** *and* ***get*** *and the test* ***pool-empty?****. A task is a tuple* (R, N, P, SG) *which consists of:*

- P *refers to a partial path in* $\mathcal{M}$ *—the so-called* ***active path****— whose continuations shall be checked for complementarity. For purposes of later proofs about our algorithms we define the active path as list of (occurrences of) literals instead of a set. (List concatenation is denoted by* $\circ$*. We write* $L \in$ P *as in case of sets.)*
- SG *refers to an occurrence of a literal in* $\mathcal{M}$ *—the current* ***subgoal****. It may temporarily have no value, which we denote by* NIL.
- R *and* N *refer to parts of the matrix—called* ***extension row*** *and* ***matrix remainder*** *respectively—through which we will extend the active path* P.

We have several subroutines which may be called during the execution of a task (R, N, P, SG)*:*

- choose_SG_column(R, SG) *non-deterministically selects a column* $\mathcal{C}^* \in$ R *which contains an occurrence* $\overline{L}^1 \Subset \mathcal{C}^*$.
  (It is called with SG $\neq$ NIL*, i.e.* SG *refers to an occurrence* L^1 *of a literal.)*
- choose_any_column(R, P) *selects a column* $C^* \in$ R *which contains a virtual column* $\widetilde{\mathcal{C}} \in \mathcal{C}^*$ *which is relevant in* R $\cup$ N $\cup$ P.
  (It is called when SG *has no current value, i.e.* SG = NIL.*)*
- choose_SG_row(C, SG) *non-deterministically selects a row* $\mathcal{R}^* \in$ C *which contains an occurrence* $\overline{L}^1 \Subset \mathcal{R}^*$.
  (It is called with SG $\neq$ NIL*, i.e.* SG *refers to an occurrence* L^1 *of a literal.)*

Let us finally mention, that apart from the non-deterministic *selection of columns and rows in the subroutines* choose_SG_column*,* choose_any_column *and* choose_SG_row*, further* non-deterministic *choices of the proof procedure* SSC *are*

- *the* ***either-or*** *choice in lines 19 and 21 and*
- *the choice of the literal occurrence* L $\in$ P *in line 20*

Definition 3 *We define a* ***configuration*** ω *as a pair* (π, S) *which represents the status of the procedure* SSC *in line 4 before 'getting' a new task.* π *is the current* ***task pool*** *and* S *is the* ***set of connections*** *constructed so far.*

Lemma 4 *The proof procedure* SSC *is correct for classical propositional logic. Given a minimal complementary matrix graph* $(\mathcal{M}, \mathfrak{S})$ *with* $\mathcal{M} \neq \emptyset$*, then the proof procedure* SSC *if applied to* $\mathcal{M}$ *will generate* $\mathfrak{S}$ *when making the right non-deterministic choices.*

5 The Virtual Column Checking Procedure

When searching for proofs in Affine Logic, we have to check acyclicity (and linearity) anyhow, and consequently, we don't need to care about complementarity of paths as long as we assure that the eventually constructed set of connections will be non-pure. In this section we will show first that the proof procedure SSC generates only non-pure sets of connections (Lemma 5). Concentrating now our attention on non-purity and forgetting about complementarity allows a refinement of the procedure SSC by dropping all tasks, which extend an active path through an already connected extension row (Lemma 6). This

```
 1.  proc SSC
 2.        R := 'given non-empty matrix M' ;   S := ∅ ;
 3.        put(R, ∅, [], NIL) ;
 4.  next: if pool-empty? then return(S) ;
 5.                          else (R, N, P, SG) := get ;
 6.        if SG ≠ NIL ∧ R is a literal then  S := S ∪ {(R, SG)} ;
 7.        if SG ≠ NIL ∧ R is not a literal then
 8.            C := choose_SG_column(R, SG) ;
 9.            N := (N ∪ R) \ {C} ;
10.            if C is a literal
11.                then put(C, N, P, SG) ;
12.                else
13.                    R := choose_SG_row(C, SG) ;
14.                    for each row R' ∈ C \ {R} put(R', N, P, NIL) ;
15.                    put(R, N, P, SG) ;
16.            fi ;
17.        fi ;
18.        if SG = NIL ∧ R is a literal then
19.                    either  /* reduction step */
20.                        choose L ∈ P with L̄ = R ;  S := S ∪ {(R, L)} ;
21.                    or
22.                        put(N, ∅, P ∘ [R], R) ;
23.        fi ;
24.        if SG = NIL ∧ R is not a literal then
25.            C := choose_any_column(R, P) ;
26.            N := (N ∪ R) \ {C} ;
27.            if C is a literal
28.                then put(C, N, P, NIL) ;
29.                else for each row R ∈ C put(R, N, P, NIL) ;
30.        fi ;
31.        goto next ;
32.  endproc
```

Figure 1: The proof procedure SSC

results in a smaller search space—without loosing non-purity, but perhaps complementarity —and will turn out later as an essential step for the generation of minimal spanning sets (Lemma 8). Adding then checks of acyclicity and linearity to this refined version of SSC leads to the proof procedure NP^{LA}_{mult} which generates only linear, acyclic, non-pure sets of connections, which according to Theorem 3 is sufficient for characterising contraction free derivability of multiplicative formulae.

Lemma 5 *If the proof procedure* SSC *applied to a non-empty matrix* $\mathcal{M}$ *returns a set* $\mathfrak{S}$ *of connections, then the matrix graph* $(\mathcal{M}, \mathfrak{S})$ *is non-pure.*

Lemma 6 *The procedure* SSC *can be refined by inserting the statement*
*'**if** not isolated* R *then goto* next ;'
after line 18 and after line 24 without loosing the non-purity of the returned set of connections.

Remark 1 There may be many top-level columns in the extension row R in case 18 and 24 in the proof of Lemma 6 and only one of them might have a virtual column which

eventually can be totally connected. With the 'test of isolatedness' to be inserted after lines 18 and 24 the repeated (and recursively stepping-down) search can be avoided, which results in exponential pruning of the search space.
Note that exponential pruning effects are achieved with ***depth-first strategy*** (see [3] for a detailed example). ■

Definition 4 (***proof procedure*** NP^{LA}_{mult}) *In order to obtain from* SSC *the proof procedure* NP^{LA}_{mult} *for multiplicative Affine Logic, we have to take measures to assure linearity and acyclicity. In addition, we want to profit from checking just virtual columns instead of paths. Unfortunately these issues interfere:*

1. *Linearity excludes reduction steps and we loose the non-deterministic choice point* $\mathtt{L} \in \mathtt{P}$ *in line 20 of* SSC. *(Lines 19–21 of the procedure* SSC *are dropped.)*
2. *Linearity and acyclicity require that connections are only made with still unconnected literals and that a new connection causes no cycle. For this reason in line 6 of* SSC *(**then**-part) the additional test*

   *'**test:*** $\mathtt{R}$ unconnected $\wedge\ (\mathcal{M}, \mathtt{S} \cup \{(\mathtt{R}, \mathtt{SG})\})$ acyclic *;'*

   will be inserted, which the non-deterministic algorithm has to pass successfully. (Note that the subgoal SG *is necessarily still unconnected—due to point 3 below—because* SG *gets its value in line 22 after possible iterations through cases 18 and 24 of* SSC, *and in case of* $\mathtt{SG} = \mathtt{NIL}$ *we never enter rows or columns which contain already connected literals.)*
3. *Finally, benefiting from Lemma 2 the task generation in the case* $\mathtt{SG} = \mathtt{NIL}$ *of* SSC *is subject to the condition that the extension row* R *is isolated. For this reason we insert after lines 18 and 24 of* SSC *the statement:*
   *'**if*** not isolated R ***then goto*** next ;'

Theorem 7 (***correctness and completeness of*** NP^{LA}_{mult}) *The proof search procedure* NP^{LA}_{mult} *is correct and complete for multiplicative Affine Logic.*

6 Minimality and Total Connectedness

When we are interested in proving theorems of multiplicative Linear Logic, we could use the proof search procedure NP^{LA}_{mult}, where we have to check in addition—according to Theorem 3—whether the matrix graphs generated by NP^{LA}_{mult} are also minimally non-pure and totally connected. In principle, such a minimality check could be carried out after a successful run of NP^{LA}_{mult}, i.e. by inserting respective tests in line 4 before ***return***(S), which must be passed successfully.
Fortunately, Lemma 8 below shows that such a final minimality checking can be avoided if we run NP^{LA}_{mult} with depth-first strategy, which is also preferable due to Remark 1. We get the proof search procedure ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$, which is the ***depth-first variant*** of NP^{LA}_{mult}, by the following modifications: The π of a configurations $\omega = (\pi, \mathtt{S})$ is now a ***task stack***, and we have operations ***push*** and ***pop*** instead of ***put*** and ***get***. We denote by $\mathrm{TOP}(\pi)$ the ***topmost task*** of a task stack π, i.e. the element which would be returned by a ***pop*** operation.

Lemma 8 *A matrix graph* $(\mathcal{M}, \mathfrak{S})$ *with* $\mathfrak{S}$ *generated by* ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$ *is minimal.*

Having got rid of the need to check minimality of the sets of connections generated by ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$, we will next try to improve the total connectedness check. To this end we prove

the following lemma which states an important fact about the 'reachability' of rows in a matrix by the procedure ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$.

Lemma 9 *Given a run of* ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$ *which generates a set of connections* $\mathfrak{S}$ *for a matrix* $\mathcal{M}$. *Then for every row* $\mathcal{R} \Subset \mathcal{M}$ *holds:*
If for the first configuration $\omega_a = (\pi_a, \mathtt{S}_a)$ *with* $\mathcal{R}$ *as extension row of its topmost task, i.e.* $\mathrm{TOP}(\pi_a) = (\mathcal{R}, \mathtt{N}_a, \mathtt{P}_a, \mathtt{SG}_a)$ *holds* $\mathtt{SG}_a = \mathtt{NIL}$, *then follows for every later configuration* $\omega_b = (\pi_b, \mathtt{S}_b)$ *with* $\mathrm{TOP}(\pi_b) = (\mathcal{R}, \mathtt{N}_b, \mathtt{P}_b, \mathtt{SG}_b)$ *that* $\mathtt{SG}_b = \mathtt{NIL}$.
As a consequence, if the row $\mathcal{R}$ *of the matrix graph* $(\mathcal{M}, \mathfrak{S})$ *is totally connected, then the row* $\mathcal{R}$ *must already be totally connected after the execution of all follow-up tasks of task* $\mathrm{TOP}(\pi_a)$.

Definition 5 (***proof procedure*** ${}^{\mathsf{df}}\mathsf{NP}^{LAT}_{mult}$) *In order to obtain from* ${}^{\mathsf{df}}\mathsf{NP}^{LA}_{mult}$ *the proof procedure* ${}^{\mathsf{df}}\mathsf{NP}^{LAT}_{mult}$ *for multiplicative Linear Logic, we only have to deal with total connectedness. Due to Lemma 9 this check can be made incrementally. We have to store respective tasks, which just contain a row* $\mathcal{R}$ *before a task with* $\mathcal{R}$ *as extension row is stored, and whose execution is just a total connectedness check. This may be done by means of the following modifications:*

- *The statement*
  '***push***$(\emptyset, \mathcal{R}, [\,], \mathtt{NIL})$;'
  is inserted before ***push***$(\mathcal{R}, \mathtt{N}, \mathtt{P}, \mathtt{NIL})$ *in lines 3, 14 and 29.*
- *The additional task processing step (for a task* $(\mathtt{R}, \mathtt{N}, \mathtt{P}, \mathtt{SG})$*) :*
  '***if*** $\mathtt{R} = \emptyset$ ***then test*** : N totally connected ;'
  is inserted before line 31.

Conclusion

Proof search in the logics treated in this paper has already attracted some interest, where the main emphasis was on Linear Logic. Most of the work carried out is based on sequent calculi and labours under the many redundancies inherent in these systems. The approach, which is most related to our work, is [4], where a proof procedure for multiplicative Linear Logic is presented. It works with a translation into classical logic. A special unification algorithm is used, which seems to be an efficient coding of the acyclicity check, which we treated here as a black box. With their procedure the generated sets of connections must pass a final check of minimality and of total connectedness, the latter being called relevance there. The main improvement of our procedure ${}^{\mathsf{df}}\mathsf{NP}^{LAT}_{mult}$ over the algorithm presented in [4] is that we got rid of the minimality check and that we can test total connectedness incrementally.

References

[1] W. Bibel. *Automated Theorem Proving.* Vieweg, 1987. (second edition).

[2] B. Fronhöfer. *The Action-as-Implication Paradigm: Formal Systems and Application*, volume 1 of *Computer Science Monographs.* CSpress, München, Germany, 1996. (revised version of Habilitationsschrift, TU München 1994).

[3] B. Fronhöfer. Proof Search in Acyclic Matrix Graphs. Technical report, Technische Universität München, 1998.

[4] Ch. Kreitz, H. Mantel, J. Otten, and S. Schmitt. Connection-based proof construction in linear logic. In W. McCune, editor, *CADE-14*, pages 207–221. Springer, LNAI 1249, 1997.